VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Blanka Chamrádová

Bakalářská práce

Vedoucí práce: Mgr. Marek Menšík, Ph.D.

Ostrava, 2021

Abstrakt

V této práci je popsán průběh absolvování individuální odborné praxe ve firmě TRAYTO. Je zaměřena primárně na popis vývoje již existujícího nástroje BiddingManager, který je využíván majiteli e-shopů pro usnadnění práce se zbožovými srovnávači Heureka.cz, Heureka.sk a Zboží.cz. Předmětem práce je tedy seznámení s vyvíjeným nástrojem, využitými technologiemi a popis několika konkrétních vypracovaných úkolů. Dále je zde zahrnut také popis samotné firmy, včetně zařazení studenta, a také celkové zhodnocení průběhu praxe s ohledem na získané znalosti.

Klíčová slova

PHP, zbožové srovnávače, optimalizace, odborná praxe

Abstract

This thesis describes the course of individual professional practice in the company TRAYTO. Its main goal is to describe development of the already existing tool called BiddingManager, which is designed to help e-shop owners with the usage of comparison shopping engines such as Heureka.cz, Heureka.sk and Zboží.cz. Therefore, the present work introduces the process of development, the technologies used and also the company itself, as well as the students position. Lastly, it summarizes students acquired knowledge and overall experience.

Keywords

PHP, comparison shopping engines, optimalization, professional practice

Poděkování

Na tomto místě bych chtěla poděkovat společnosti TRAYTO za možnost absolvovat odbornou praxi a stejně tak všem mým kolegům, na které jsem se vždy mohla obrátit se svými dotazy. Zejména pak děkuji Bc. Martinu Mohlerovi, který mi poskytl značné množství znalostí využitých při absolvování této praxe.

Obsah

Seznam použitých zkratek a symbolů

AMQP	–	Advanced Message Queuing Protocol
API	–	Application Programming Interface
CPC	–	Cost Per Click
DBMS	–	Database Managment System
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
JSON	–	JavaScript Object Notation
PHP	–	PHP: Hypertext Preprocessor
REST	–	Representational State Transfer
RSS	–	Rich Site Summary
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Předmětem této bakalářské práce je mé působení ve firmě TRAYTO a vývoj nástroje s názvem BiddingManager. Absolvování individuální praxe jsem si vybrala, abych své teoretické znalosti z vysoké školy doplnila i o zkušenosti z praxe, protože dlouhou dobu pro mne byla práce ve firmě v podstatě nepředstavitelná. Za krátkou dobu působení u TRAYTO jsem ale nabyla mnoha dovedností, mezi nimi například jak pracovat v týmu nebo jak probíhá vývoj aplikací větších rozměrů.

Druhá kapitola se zabývá popisem firmy TRAYTO, mým zařazením ve firmě a postupem vývoje projektů ve firmě. Třetí kapitola seznamuje s vyvíjeným nástrojem BiddingManager a technologiemi využívanými k jeho vývoji. Ve čtvrté kapitole jsou shrnuty jednotlivé úkoly, na kterých jsem během praxe pracovala a v páté, šesté a sedmé je popsáno mé vlastní řešení zadaných úkolů. Poslední kapitola obsahuje zhodnocení znalostí načerpaných během studia, které jsem využila během absolvování praxe, ale také zhodnocení znalostí, které mi chyběly.

Kapitola 2

Představení společnosti

Společnost TRAYTO je firma zaměřená na práci s produktovými daty, těžbu online dat a automatizaci procesu přihazování ceny za proklik (nazývaného bidding) na zbožových srovnávačích. Vznikla jako ryze česká společnost roku 2014 v Ostravě. Dříve mohla být známa pod názvem BiddingTools Group, od doby vstupu nového investora v roce 2019 však BiddingTools Group vystupuje jako TRAYTO. Jedná se o středně velkou firmu, která aktuálně zaměstnává okolo třiceti osob. Nezabývá se pouze vývojem softwaru, ale také poradenstvím v oblasti e-commerce souvisejícím se zbožovými srovnávači. Hlavním zaměřením vývoje jsou právě vlastní produkty a nástroje. Kromě nástroje BiddingManager, popisovaném v této práci, nabízí například nástroj sloužící k editaci XML feedů, Xemel nebo také individuální řešení pro klienty ohledně zpracování dat. Nedávno se vlastníkem společnosti TRAYTO stala skupina Packeta, díky čemuž se může společnost soustředit také na expanzi na zahraniční trh, jak sama uvádí na svém webu [1]. Společnost Packeta mimo jiné vlastní také společnost Zásilkovna, se kterou společnost TRAYTO začala pracovat na společných projektech.

2.1 Zařazení studenta ve firmě

Celkem ve firmě pracuje 9 vývojářů a dle potřeby dochází ke spolupráci s externími týmy, přičemž je firma nositelem organizace a spolupráce mezi nimi. Za těchto podmínek ale není prostor pro přílišnou specializaci jednotlivých členů týmu, což vede k tomu, že si většina vývojářů projde celým procesem vývoje od analýzy až k testování či nasazení. Tým je rozdělen na základě projektů do menších skupin po dvou až třech vývojářích a v případě potřeby dochází k přesunům vývojářů mezi projekty (vyjma projektů vycházejících ze spolupráce se společností Zásilkovna). Každý vývojář by tedy měl mít alespoň základní přehled o ostatních projektech a měl by být schopen odbavit úkoly týkající se všech firemních nástrojů, jelikož kvůli malému počtu vývojářů v týmech nelze zajistit, že každý z projektů bude vždy pokryt dostatečným počtem vývojářů. K těmto přesunům tedy dochází zejména v případech, kdy je potřeba implementovat rozsáhlejší změny na projektech nebo při vzniku nového projektu, kdy se jedná o přesun dlouhodobý. Většina vyvíjených nástrojů

však pracuje se zbožovými srovnávači a jejich daty a jsou tedy založeny na podobných principech, což vede k tomu, že orientace v ostatních projektech není pro vývojáře tak složitá, jelikož jsou již seznámeni s fungováním zbožových srovnávačů.

Já jsem ve firmě absolvovala praxi na pozici software developer (vývojář) junior a mým primárním projektem byl BiddingManager (blíže popsán dále v kapitole 3), jehož tým se skládal ze tří vývojářů včetně mě a jedné produktové specialistky. Produktový specialista je osoba, která dohlíží na kvalitu a marketingové strategie konkrétního produktu a na základě zpracování zpětné vazby zákazníků nebo vlastních analýz zadává úkoly danému týmu vývojářů.

Po celou dobu jsem pracovala pod dohledem lead developera, který byl pověřen konzultantem této práce jako můj mentor ve firmě.

2.2 Způsob vývoje

Ve firmě se aplikují metody agilního vývoje, konkrétně metodika Scrum, což znamená způsob vývoje založený na iteracích, kdy výsledkem každé iterace je hotový produkt nebo jeho část. Výhoda oproti sekvenčnímu přístupu spočívá v možnosti měnit požadavky a zadání. Iteracemi ve firmě jsou týdenní sprinty a každý týden na začátku sprintu probíhají předběžná plánování pro jednotlivé projekty, na kterých se rozdělí úkoly mezi vývojáře. Úkoly zadává převážně produktový specialista, případně osoba na pozici lead developer s tím, že je kladen důraz na podrobnou specifikaci. Všechny zadávané úkoly se nejprve ukládají do „backlogu“, což je seznam všech plánovaných a dosud nevyřízených úkolů. Je potřeba, aby před zařazením úkolu do sprintu vývojáři určili přibližné časové náročnosti úkolu, které pomáhají produktovému specialistovi naplánovat množství úkolů ve sprintu.

Průběh vývoje se kontroluje denně na cca 5 minutové schůzi, takzvaném „StandUpu“, na kterém každý z vývojářů stručně informuje, jak na zadaných úkolech pokročil, na jaké problémy narazil a na čem bude dále pokračovat. Tímto získávají o vývoji úkolů přehled nejen ostatní vývojáři, ale hlavně produktoví specialisté, pro které je důležité, zda bude úkol vyřešen včas. Osobně si myslím, že tato forma komunikace mezi celým týmem je velice prospěšná, jelikož produktoví specialisté jsou průběžně informováni o tom, kdy mají očekávat dodání úkolů na kontrolu, a vývojáři jsou obeznámeni se vším, co na projektech přibývá.

Každý hotový úkol by měl projít přes code review, což je kontrola kódu jiným vývojářem. Úkoly ke kontrole si mezi sebe vývojáři rozdělují také na „StandUpech“. V případě, že budou v kódu nalezeny chyby, autor je informován a musí je opravit. Těmito chybami může být nedodržování standardů kódu, nesrozumitelný kód nebo také neefektivní řešení a chybný návrh řešení. Po úspěšném code review úkol pokračuje na testování a byznys validaci produktovým specialistou, kdy produktový specialista kontroluje, zda vypracovaný úkol funguje, jak má, případně testuje vypracování úkolu z pohledu zákazníka. V případě úkolů zadaných osobou na pozici lead developer, které nemusí být produktový specialista schopný otestovat, může tuto validaci provádět sama osoba na pozici lead developera. Až po úspěšné validaci může být vypracovaný úkol nasazen na produkci.

Pro tento způsob vývoje jsou dostupné aplikace na řízení projektů jako například Jira, který firma začala před nedávnem používat. Umožňuje plánování sprintů oddělených dle jednotlivých projektů nebo sledování fází vývoje od implementace, přes code review až po nasazení. Dále Jira nabízí také propojení s Git repozitáři služby Bitbucket, kdy automaticky přiřazuje specifikaci úkolů k jejich commitu, což je užitečné zejména pro zpětné vyhledávání specifikací.

Kapitola 3

BiddingManager

Biddingmanager je webový nástroj umožňující klientům jednodušší bidding na zbožových srovnávačích Heureka.cz, Heureka.sk a Zboží.cz. Bidding je proces zvyšování ceny za proklik (CPC) produktu, což vede k jeho zviditelnění ve srovnávači na takzvaných biddovaných pozicích. Kromě možnosti nastavení biddingu nabízí BiddingManager také přehledné statistiky o produktech či kategoriích, ale i možnost autobiddingu, kdy uživatel zadá pozici, na které se má produkt ve zbožovém srovnávači zobrazit, a BiddingManager automaticky počítá potřebnou CPC. Uživatelé mají možnost spravovat pod jedním účtem na BiddingManageru účty svých obchodů na všech podporovaných srovnávačích, k jednomu obchodu tak můžeme přiřadit až tři účty srovnávačů. Více informací o biddingu a samotném nástroji BiddingManager je k nalezení na webu nástroje [2].

3.1 Vznik nástroje

Původně pro interní účely marketingového oddělení vznikl doplňkový nástroj pro agregaci statistik biddingu, aby firma získala lepší přehled o funkčnosti služby BiddingTools a automatizaci procesů marketingového oddělení. Tento interní nástroj byl vyvinut bez předchozí podrobné analýzy členem marketingového oddělení, který se okrajově zajímal také o programování. Služba BiddingTools [3] zajišťuje pro větší e-shopy komplexní a exkluzivní správu na zbožových srovnávačích. Jelikož je služba exkluzivní a tím pádem omezená pro určitý počet konkurenčních e-shopů zaměřených na stejnou kategorii zboží, vznikla také poptávka po nástroji pro automatizaci biddingu ze strany menších a středních e-shopů. Pro rychlé uspokojení poptávky se firma rozhodla využít výše zmíněný interní nástroj a vytvořit z něj biddovací nástroj s administračním rozhraním pro snadnější bidding na zbožových srovnávačích, který pojmenovala BiddingManager. V návaznosti na jeho komerční využití byl poté předán týmu programátorů pro další vývoj a údržbu. V rámci tohoto postupu vyvstaly problémy spojené s absencí analýzy a návrhu původního softwaru, které firma musí zpětně řešit. Průběžně tedy firma pracuje na vylepšení návrhu stávajícího kódu a dalším rozvoji.

3.2 Data zbožových srovnávačů

Aby mohl BiddingManager správně fungovat, potřebuje denně získávat mnoho dat ze zbožových srovnávačů. Vzhledem k jeho zaměření na bidding jsou klíčovým prvkem informace o nabízených produktech, ty e-shopy poskytují v podobě URL adresy s XML feedem, v něm se nachází data o ceně, výrobci, naskladnění a dopravě nebo také kategorii každého produktu. Ve srovnávačích Heureka.cz a Heureka.sk si uživatelé mohou stáhnout dokument, takzvaný sortiment report, informující o tom, jak si jednotlivé produkty vedou oproti konkurenčním e-shopům, například počet konkurentů nebo nejnižší cenu na trhu. Tento dokument poté BiddingManager využívá ve svých statistikách. Je také třeba znát informace o produktových kartách. Produktové karty slouží pro návštěvníky zbožových srovnávačů a obsahují popis vybraného produktu, srovnání nabídek všech e-shopů, které jej na zbožovém srovnávači inzerují, obrázky produktu a podobně. Informace z produktových karet se jednoduše dají získat přímo přes rozhraní pro programování aplikací (API), které zbožové srovnávače nabízejí. Protože tato API jsou využívána napříč více projekty, jsou vyvinuty adaptéry zapouzdřující komunikaci s nimi, což jednak zabraňuje duplikovanému kódu, ale také ulehčuje práci vývojářům.

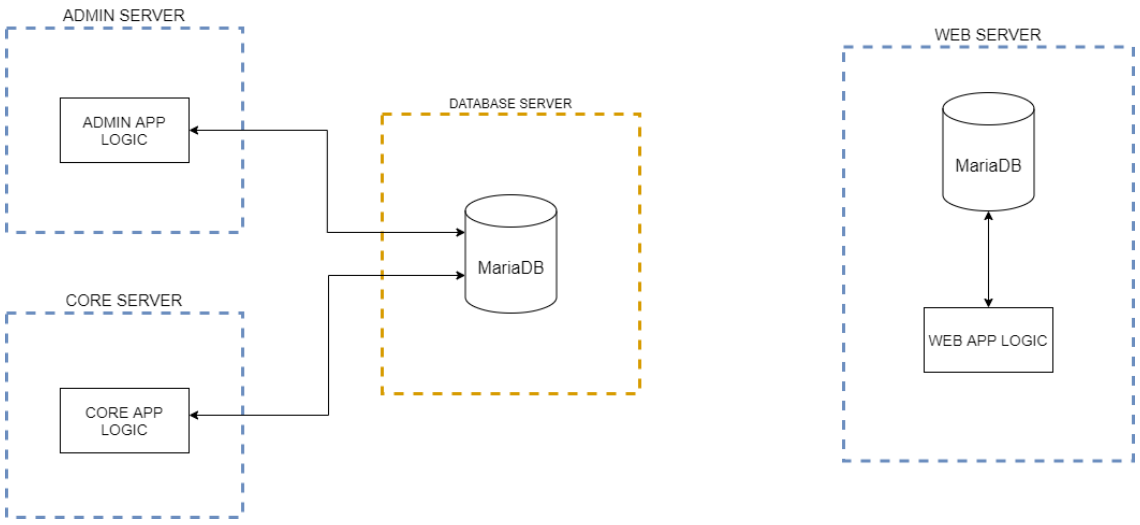
Stále však existují informace, které zbožové srovnávače přímo neposkytují a je tedy potřeba získávat je přímo ze stránek administrace jednotlivých účtů na srovnávačích za využití HTTP klientů na tyto stránky. Jedná se o informace o výši kreditu pro bidding a dále také o certifikaci e-shopu na Heureka.cz a Heureka.sk. Certifikát e-shopu je potřeba znát, jelikož bez něj není možné na těchto zbožových srovnávačích nastavovat CPC.

3.3 Architektura

BiddingManager je rozdělen na více částí, zaměřených na specifickou činnost (například uživatelské rozhraní, správa notifikací nebo fakturace). Každá část běží na vlastním Apache serveru a navzájem spolu nekomunikují, ale jsou připojeny na stejný databázový server s databázovým systémem MariaDB. Databáze na serveru jsou dále rozděleny na databáze pro každý ze tří zbožových srovnávačů (Heureka.cz, Heureka.sk, Zboží.cz), obecnou databázi s daty o uživatelských účtech, databázi zprostředkovávající spouštění skriptů (dále popsáno v kapitole 6) a databázi sloužící pro uživatelské notifikace.

Kromě samotného nástroje na biddování má BiddingManager i svůj web oddělený od biddovacího nástoje, tím pádem běží také na svém vlastním serveru společně se svým vlastním databázovým systémem MariaDB. Architektura je znázorněna na obrázku 3.1

Dále existují také servery včetně databáze pro testování implementovaných úkolů před nasazením do produkce. Obsahují přesnou kopii nástroje a každý úkol je nejdříve nahrán na tento server, kde produktový specialista nebo lead developer mohou provést byznys validaci.



Obrázek 3.1: Architektura serverů BiddingManageru

Část nazývaná admin zprostředkovává webové rozhraní pro klienty, ve kterém mohou spravovat své nastavení a zobrazovat požadované statistiky a přehledy. Uživatelská nastavení jsou ukládána do databáze, aby s nimi mohl nástroj dále pracovat.

Další podstatnou částí je core část, která zajišťuje samotné chování a logiku nástroje. Toho je docíleno za pomoci spouštění takzvaných tasků.

Task

Jako tasky v nástroji BiddingManager jsou označovány skripty v core části, které zajišťují základní funkčnost nástroje. Zprostředkovávají zejména výměnu dat mezi nástrojem a zbožovými srovnávači, ale také například výpočty pro určení doporučené CPC. Těchto tasků existuje dohromady 10 typů, jsou však implementovány pro každý ze zbožových srovnávačů zvlášť (Heureka.cz, Heureka.sk nebo Zboží.cz), jelikož způsoby komunikace se srovnávači jsou odlišné. Všechny tyto tasky jsou spouštěny automaticky v určité časy za využití cron výrazu pro každý z aktivních účtů na srovnávačích zvlášť. Mezi nejdůležitější činnosti prováděné tasky patří například:

- Spuštění generování sortiment reportu na Heureka.cz a Heureka.sk
- Autobidding
- Doporučení produktů pro autobidding
- Výpočet optimální ceny CPC pro uživatele

3.4 Využité technologie

Téměř denně se vývojáři setkávají s několika technologiemi, nástroji či knihovnami souvisejícími s jazykem PHP. Díky nim je vývoj značně usnadněn a většina z nich je využívána napříč všemi projekty, nejen v BiddingManageru. Firma dbá na neustálý posun technologií vpřed, takže stále přibývají další a další.

PHP

PHP [4] je univerzální skriptovací jazyk hodící se především na tvorbu webových aplikací. Byl vytvořen v jazyku C roku 1994 a jeho původní funkcí mělo být vkládání C kódu do HTML dokumentů. Tato vize se však velmi liší jeho od současného stavu, kdy je využíván jako plnohodnotný programovací jazyk, ve kterém lze vytvářet jak webové, tak konzolové aplikace. Informace o vzniku jazyka jsem čerpala v knize [5]. Aktuální verzí je verze 8 vydaná na konci roku 2020.

PHP je ve většině částí BiddingManageru využíván převážně ve verzi 7.4, v historických částech se můžeme setkat s verzí 5.6, která se ale postupně refaktoruje. Aktuálně je PHP pro firmu dostačující jako hlavní programovací jazyk.

MariaDB

MariaDB je open source systém řízení báze dat (DBMS) nabízející relační databáze, to jsou databáze umožňující rozdělení dat do oddělených tabulek. K přístupu do databázi používá jazyk SQL. Vychází z DBMS MySQL a jeho vývoj je veden částí původních vývojářů MySQL, vytvořen byl z důvodu udržení otevřenosti softwaru, o kterou MySQL přišla. Zakládá se právě na otevřenosti, ale také na výkonnosti a stabilitě, jak je uvedeno na oficiálním webu [6].

Pro správu databáze pomocí webového rozhraní firma využívá nástroj Adminer.

Composer

Composer je nástroj pro správu závislostí v PHP, slouží k instalaci, aktualizaci a správě využívaných knihoven. Dává nám možnost provádět tyto akce přímo přes terminál, ale také jsou všechny informace o závislostech uvedeny přehledně v jednom souboru *composer.json*, kterým se nástoj řídí při instalaci. Je schopen udržovat instalované knihovny v takových verzích, aby byly navzájem kompatibilní, a stejně tak aby byly kompatibilní s verzí PHP v projektu. Základy práce s tímto nástrojem jsou popsány v knize [7]. Hlavním repozitářem, kde můžeme vyhledávat knihovny je Packagist, nemusíme se však spoléhat pouze na ty publikované jinými vývojáři, Composer umožňuje také přidávání vlastních repozitářů.

Do BiddingManageru byly v průběhu vývoje přidány desítky knihoven, které musí být pro jeho funkčnost nainstalovány. Díky dostupnosti souboru *composer.json* je jejich instalace otázkou jednoho příkazu.

Git

Git je open source verzovací systém poskytující možnost sledování změn v kódu jednotlivých repozitářů reprezentujících projekty. Každá změna je uložena jako commit, který obsahuje informace o provedených změnách, autorovi změn a času změny. Jedná se o nezbytný nástroj pro vývoj v týmu, jelikož na projektech často pracuje více vývojářů a každý z nich pracuje na svých vlastních úkolech. Ke každému úkolu je vytvořen samostatný branch, ve kterém se změny implementují, a po dokončení je nahrán do master branche, z něhož výchází všechny nové branche. Více informací lze nalézt v knize [8].

S využitím systému Git se váže i využití služby Bitbucket, která poskytuje webové rozhraní pro správu prováděných změn. Konkrétně Bitbucket byl firmou zvolen kvůli jeho návaznosti na software Jira, což umožňuje propojení zadaných úkolů s projekty a jejich commity.

Nette

Nette je PHP framework poskytující Model View Controller (MVC) architekturu, ta zajišťuje logické oddělení databázových operací a front-end části za pomoci Presenteru propojujícího tyto dvě vrstvy. Pro vykreslování dat se využívá šablonovací jazyk latte.

Ve firmě je tento framework nebo některé jeho samostatné komponenty využíván v mnoha projektech. BiddingManager celý framework používá na svém webu a v core části slouží k odesílání emailů samostatný balíček Nette Mail.

Guzzle

Guzzle je knihovna fungující jako HTTP klient pro PHP, který zprostředkovává odesílání HTTP požadavků. Uživatelům je dostupné jednoduché rozhraní s mnoha možnostmi sestavení požadavků. Veškeré informace o této knihovně jsem čerpala z její oficiální dokumentace [9].

Pro BiddingManager se jedná o velice důležitou knihovnu, jelikož odesílání HTTP požadavků je v našem případě na denním pořádku, ať už se jedná o zjišťování informací z webového rozhraní zbožových srovnávačů nebo komunikaci s API. Samotné PHP poskytuje knihovnu pro tuto komunikaci, cURL, firma však volí raději právě Guzzle, jelikož práce s ním je přehlednější a je obohacen o některé funkce, jako například rozlišení synchronních a asynchronních požadavků.

Kapitola 4

Seznam přidělených úkolů

Seznámení se s pracovními postupy a používanými technologiemi

Jelikož se jedná o mou první práci jako software developer a s jazykem PHP jsem měla před nástupem do firmy minimální zkušenosti, musela jsem si ze začátku osvojit způsob vývoje, a to převážně prací na menších úkolech, díky kterým jsem se seznámila s většinou používaných technologií ve firmě. Během prvního měsíce a půl jsem si tak vyzkoušela práci na všech možných typech aplikací, od interních programů, přes práci na klientovi zprostředkujícím komunikaci s API Zboží.cz, až po vytvoření projektu ke zpracování dat pro klienta firmy. Tyto počáteční úkoly byly poměrně rozmanité, jelikož jsem tímto způsobem mohla nahlédnout do implementace všech typů vyvíjených aplikací. Také jsem se díky nim seznámila s potřebnými informacemi ohledně fungování zbožových srovnávačů.

Časová náročnost 120 hodin.

Práce na firemních webech

Díky práci na webových stránkách jsem se seznámila se způsobem tvorby webových aplikací ve firmě, konkrétně pak s frameworkem Nette a celkově s MVC architekturou, kterou framework poskytuje. Pracovala jsem hlavně na back-end vývoji webových stránek, neminuly mě však ani úpravy front-end prezentace webů. Více o práci na firemních webech popisuji v kapitole 5.

Časová náročnost 80 hodin.

Optimalizace nástroje BiddingManager

Jak již bylo zmíněno v části 3.1, firma pracuje na vylepšení a dalším rozvoji nástroje BiddingManager. V kapitole 6 detailně popisuji jednotlivé úkoly spojené s vylepšením nástroje a jejich řešení.

Časová náročnost 180 hodin.

Změny v obchodním modelu nástroje BiddingManager

K 1. 2. 2021 se firma rozhodla provést změny v obchodní modelu nástroje BiddingManager. Změny se týkaly především způsobu určování ceny za nástroj a velikosti uživatelského balíku pro bidding, z čehož vyplynuly nutné úpravy v back-end i front-end částech nástroje. Má práce na tomto úkolu bude dále popsána v kapitole 7.

Časová náročnost 20 hodin.

Kapitola 5

Práce na firemních webech

5.1 Vytvoření RSS kanálu

V rámci tohoto úkolu mi bylo zadáno zpřístupnit články na firemním blogu BiddingBlog pro RSS (Rich Site Summary) čtečky vytvořením RSS kanálu (feedu). RSS feed obsahuje ve formátu XML informace o dostupných článcích, například datum zveřejnění, odkaz na článek nebo obrázek.

RSS kanál vystupuje jako samostatná stránka na blogu, bylo tedy potřeba vytvořit takzvaný Presenter s názvem *RSSPresenter*, který bude jeho vypisování zprostředkovávat. Presentery ve frameworku Nette slouží ke zpracování požadavků ohledně dat, která se mají zobrazit v šabloně (vrstva View), nebo naopak akcím prováděným v úložišti (vrstva Model). Zabraňují tak propojení těchto dvou vrstev.

K vytvoření XML řetězce, který bude reprezentovat RSS feed jsem vytvořila metodu *generateRssFeed* s návratovým typem string. Všechna data, která bylo potřeba do RSS feedu zahrnout, bylo možné nalézt v databázové tabulce *article*. Tato data se dala získat za využití již existující třídy *ArticleRepository* a její metody *getArticles*. Stačilo tedy pro každý z článků přidat XML element do výsledného řetězce. Pro vytvoření XML řetězce byla využita knihovna *bhaktaraz/php-rss-generator*, která nabízí metody pro přidávání jednotlivých položek RSS feedu. Do RSS feedu je možné začlenit také obrázky, jejich odkazy bylo možné sestavit díky id článku.

Zobrazení RSS feedu po načtení stránky jsem zajistila vypsáním řetězce navráceného metodou *generateRssFeed* v metodě *renderDefault* třídy *RSSPresenter*. Nyní již bylo možné zobrazit RSS feed na svém vlastním odkazu. V rámci lepší dostupnosti feedu pro potencionální čtenáře jsem však musela navíc do zdrojového kódu hlavní stránky přidat HTML link (zobrazen ve výpisu 5.1), který informoval o dostupnosti feedu na stránce. Poté již na existující RSS feed mohly upozorňovat například samy prohlížeče.

Bohužel na konci roku 2020 BiddingBlog zanikl, jelikož došlo k přesunu obsahu blogu pod obecný firemní blog, který se zaměřuje na širší spektrum témat včetně informací o biddingu.

```
<link rel="alternate" title="BiddingBlog.cz" href="{$basePath}/rss/" type="
    application/rss+xml">
```

Výpis 5.1: Link pro přidání RSS feedu

5.2 Úpravy e-mailů s novinkami

Na základě rozhodnutí firmy mělo dojít ke změně ve způsobů odesílání e-mailů s novinkami za využití platformy Ecomail. Tato platforma by zjednodušila práci marketingových specialistů, jelikož by získali přehledné rozhraní pro správu odesílání e-mailů a navíc také přehled o statistikách již odeslaných e-mailů.

K využití platformy bylo potřeba upravit přihlášení k odběru novinek takovým způsobem, aby po zadání e-mailu do formuláře došlo k jeho přidání do seznamu v platformě Ecomail. K těmto činnostem nabízí Ecomail své vlastní API.

Komunikaci s API jsem zapouzdřila do třídy *EcomailApi*, která poskytuje metody *addSubsriber* (výpis 5.2) a *isSubscribed*. Při přidávání a ověřování odběrů jsem musela brát ohled na skutečnost, že platforma může být využívána pro více firemních nástrojů najednou. E-maily odběratelů proto jsou označeny štítkem („tagem“), který specifikuje, ke kterému nástroji e-mail patří.

```
public function addSubscriber(string $email) : Response
{
      $headers = ['key' => $this->key];
      $data = [
         'subscriber_data' => [
            'email' => $email,
            'tags' => [self::ecomailTag]
         ],
         'trigger_autoresponders' => false,
         'update_existing' => true,
         'resubscribe' => false
      ];
      $postFields = \GuzzleHttp\json_encode($data);

      $response = $this->client->request('POST', 'lists/' . $this->listId . '/
          subscribe', ['headers' => $headers, 'body' => $postFields]);
      return $response;
}
```

Výpis 5.2: Metoda pro přidání e-mailu odběratele

Po odeslání formuláře nejprve pomocí metody *isSubscribed* dojde ke kontrole, zda již zadaný e-mail není zaregistrován. Pokud ano, bude přesměrován na stránku informující o této skutečnosti, v opačném případě dojde k uložení e-mailu za pomoci metody *addSubscriber* a uživateli bude již přes platformu Ecomail automaticky odeslán potvrzovací e-mail.

5.3 Ostatní úkoly

Dalším z mých úkolů bylo zvýšit verzi Nette frameworku z 2.4 do 3.0 na firemních webech. Tento přechod na vyšší verzi byl nutný, jelikož byla ve verzi 2.4 objevena bezpečnostní chyba umožňující potencionálním útočníkům spouštět PHP kód na stránkách. Ve verzi 3.0 však došlo k několika změnám, na základě kterých byly potřeby drobné úpravy kódu.

Dále jsem pracovala na refaktorování (vylepšení struktury kódu) firemního blogu, kdy bylo potřeba rozdělit Presentery a šablony do menších logických částí. Cílem bylo vytvořit z komponenty *HomepagePresenter* několik méně obsáhlých komponent a odpovídající šablony.

Navíc jsem také webu nástroje BiddingManager vytvářela novou patičku, navigaci a další úpravy ohledně rozložení stránek.

Kapitola 6

Optimalizace nástroje BiddingManager

V této kapitole popisuji řešení úkolů spojených s vylepšením stávajícího kódu nástroje BiddingManager, které souvisejí s netradičním postupem při vzniku nástoje popisovaným v části 3.1.

6.1 Úpravy zpracování tasků

Před úpravami zpracování tasků fungovalo za využití databáze. Informace o tasku, který se měl provést, byly ukládány do databázové tabulky. Následně tyto informace odebíral skript běžící v nekonečném cyklu, takzvaný worker, který na základě získaných informací jednotlivé tasky spouštěl. V tabulce vedle informací nutných pro spuštění tasku, jako je jeho název, id účtu srovnávače nebo priorita, byla evidována také délka běhu a informace o úspěšnosti jeho provedení (všechny informace jsou zobrazeny v tabulce 6.1).

Firma se rozhodla pro přidání nového virtuálního serveru pro zpracování tasků za účelem navýšení výkonu. Funkčnost nového serveru bude stejná jako u původního core serveru, zpracuje se tedy dvojnásobné množství tasků najednou. S výše popisovanou implementací by ovšem bylo obtížné zajistit, aby na obou serverech nedocházelo ke zpracovávání stejných tasků najednou, protože workery by přistupovaly do databáze paralelně, a tak by mohlo dojít k nechtěnému odbavování stejného tasku vícekrát. V návaznosti na tyto změny mi byl zadán úkol implementovat nový způsob zpracování tasků.

6.1.1 Požadavky na zpracování tasků

Hlavním požadavkem pro zpracování tasků byla správná distribuce mezi oběma core servery. K řešení tohoto požadavku se firma rozhodla pro využití fronty zpráv. Fronta zpráv, neboli message queue, je technologie umožňující komunikaci mezi aplikacemi za využití odesílání zpráv pomocí asynchronního komunikačního protokolu.

Požadováno bylo dále zachování prioritizace jednotlivých tasků. Vhodné bylo zaměřit se také na ošetření duplikovaných zpráv. Pokud by totiž fronta byla plněna zprávami týkajícími se tasku, který

tasks

Název	Datový typ
uuid	integer
name	varchar(256)
sid	integer
is_executed	integer
is_finished	integer
is_error	integer
output	varchar(256)
priority	integer
retries	integer
time_created	timestamp
time_executed	timestamp
time_finished	timestamp

Tabulka 6.1: Tabulka *tasks*

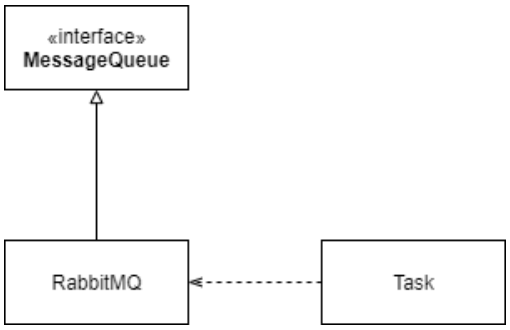
se již ve frontě nachází, mohlo by docházet ke zpracování totožných tasků několikrát za sebou. Tento jev není žádoucí například u tasků jako je autobidding, protože by docházelo k přepočtu CPC na základě již neaktuálních pozic dle biddingu na produktových kartách. Nové CPC se do zbožových srovnávačů totiž propisuje přibližně jednou za dvě hodiny. Proto do fronty nebudou zařazovány žádné duplikované zprávy.

Jako doporučený typ fronty pro tento úkol zvolila firma RabbitMQ, před samotnou implementací jsem však měla za úkol zjistit, zda poskytuje možnost řešení všech specifikovaných požadavků.

RabbitMQ

RabbitMQ je open source message broker. Na rozdíl od prosté fronty zpráv message broker zvládá informace ve svých frontách navíc různě zpracovávat, jedná se tedy o rozšíření fronty zpráv. Umožňuje rozsáhlá a pro nás vhodná nastavení, některá přímo ve svém základu, jiná dostupná v doplňkových modulech. Informace o RabbitMQ jsem čerpala v knize [10]. Další výhodou je možnost vlastního hostingu serveru, na kterém RabbitMQ běží, a velký limit pro velikost zpráv.

Pro využití RabbitMQ bude tedy potřeba určit server, na kterém software RabbitMQ poběží, a následně jej nainstalovat společně s balíčkem pro ošetření duplikovaných zpráv. Dále bude nainstalován takzvaný *management plugin*, což je balíček nabízející monitorování a správu běžícího serveru. Díky tomuto balíčku bude možný přístup k webovému rozhraní, kde budou zobrazeny například informace o existující frontě nebo množství zpráv v ní. Tímto budou pokryty všechny požadavky pro zpracování.



Obrázek 6.1: Vztah mezi třídami při implementaci RabbitMQ

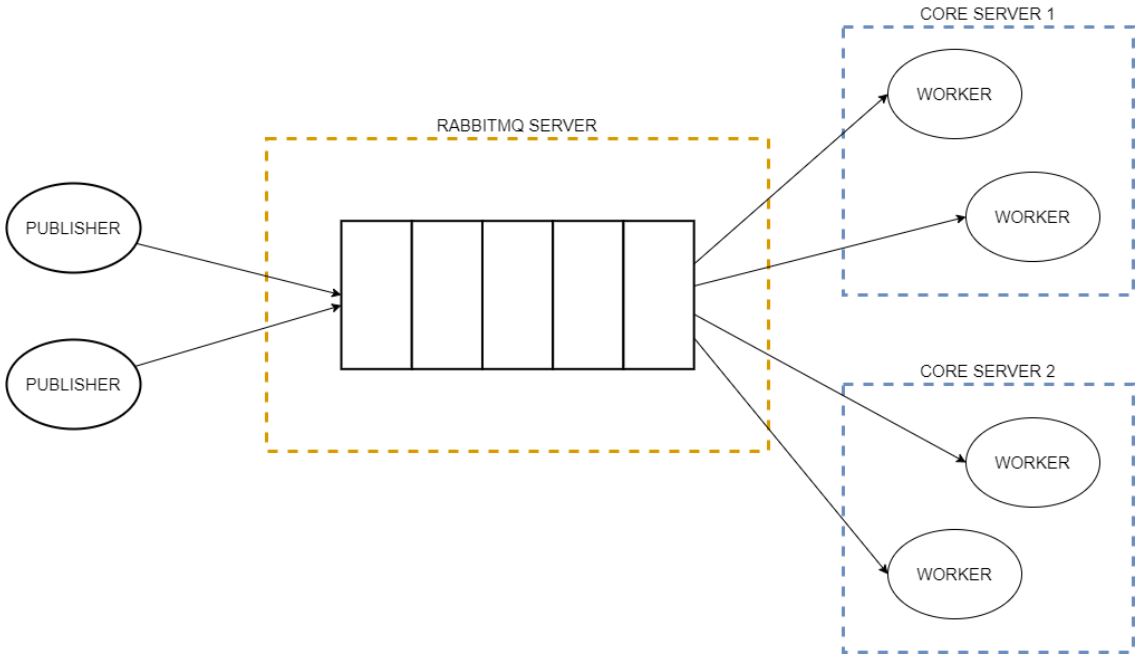
6.1.2 Návrh zpracování tasků

Jako předpis pro práci s frontami zpráv bude vytvořeno rozhraní *MessageQueue*. Obecnými metodami pro všechny typy front zpráv budou publikování a odebírání zpráv. Z tohoto rozhraní bude vycházet třída *RabbitMQ*. V místech, kde se informace o tasku vkládaly do databáze, bude nově připojení do databáze nahrazeno odesláním zprávy do fronty. Ke zpracování odesílaných informací a následnému odeslání pomocí třídy *RabbitMQ* bude využita třída *Task*, která bude reprezentovat takzvaného *Publishera*. Vztah mezi třídami je popsán v doménovém modelu na obrázku 6.1.

Tasky z fronty bude odebírat upravený worker reprezentující tzv. *Consumera*. Těchto workerů bude moci paralelně běžet několik současně a nově i na více serverech současně. RabbitMQ se postará o korektní distribuci mezi jednotlivé workery. Schéma fungování je popsáno na obrázku 6.2.

6.1.3 Řešení zpracování tasků

Prvním krokem bylo vytvoření rozhraní *MessageQueue*, jež udává metody, které se pro práci s frontou zpráv musí implementovat. Z tohoto rozhraní již pak mohla vycházet třída *RabbitMQ* pro zprostředkování komunikace s RabbitMQ. V konstruktoru bude třída požadovat údaje o serveru, přihlašovacích údajích k RabbitMQ serveru a názvu fronty. K připojení ke frontě dojde ihned v konstruktoru této třídy, nejdříve se však musí dojít k navázání spojení se serverem, na kterém běží RabbitMQ, za využití Advanced Message Queuing Protocol (AMQP). AMQP je protokol sloužící právě pro využití message brokerů a vyměňování zpráv mezi nimi a klientskými aplikacemi. Jedná se o hlavní protokol, který RabbitMQ podporuje, ale je možné využití i jiných protokolů, například Message Queuing Telemetry Transport (MQTT). Zvolila jsem AMQP, jelikož je práce s ním v rámci RabbitMQ doporučena a nejlépe zdokumentována. Způsob využití protokolů je dále popsán v knize [10]. Po navázání spojení může dojít k vytvoření kanálu, který bude zprostředkovávat veškerou komunikaci s frontou. Kanály eliminují potřebu vytvářet neustále nová spojení, čímž snižují využití systémových prostředků. Komunikaci za pomoci AMQP nelze provádět bez využití těchto kanálů. Frontu, se kterou bude kanál pracovat, se identifikuje pomocí jejího jména. Při deklaraci fronty je



Obrázek 6.2: Schéma fungování RabbitMQ v BiddingManageru

potřeba také definovat její chování. S ohledem na zadané požadavky jsem povolila využití prioritizace zpráv a odstranění duplikátů, ale také jsem specifikovala odolnost fronty. Odolností je v tomto případě myšleno uchování zpráv i po vypnutí RabbitMQ serveru. Metody třídy *RabbitMQ publish* a *subscribe* jsem doplnila o čtení a odebírání zpráv z vytvořeného kanálu.

Jelikož jsou třídě *RabbitMQ* v konstruktoru předávány informace, které budou obsaženy v konfiguračním souboru, musí před jejím vytvořením dojít k jejich načtení. K tomu bylo možné využít již existující třídu *Config*, která dokáže data z konfiguračních souborů číst. K řešení této závislosti se poté nabízelo využít návrhový vzor *Dependency injection*. V projektu již existovala třída *Container*, která sloužila k vytváření závislých objektů, musela jsem do ní tedy zaregistovat třídu *RabbitMQ*. Poté již bylo možné vytvářet instanci třídy *RabbitMQ* bez specifikace údajů z konfiguračního souboru, jak je znázorněno ve výpisu 6.1.

```
$rabbitMQ=$container->get(RabbitMQ::class);
```

Výpis 6.1: Ukázka využití třídy Container

K odesílání do fronty bude docházet v kódu na více místech, na základě návrhu jsem proto vytvořila třídu *Task* umožňující vkládání informací o tasku do fronty. Ta obsahuje jedinou statickou metodu, která bude naplňovat frontu dle zadaných parametrů. Těmi jsou:

- Název tasku
- Id účtu srovnávače v databázi
- Název databáze, ve které jsou uloženy data ke konkrétnímu účtu srovnávače
- Typ zbožového srovnávače (Heureka.cz, Heureka.sk, Zboží.cz)
- Priorita

Dále stačilo jen vytvořit AMQP zprávu. Samotné tělo zprávy je vytvořeno převedením zadaných parametrů do formátu JSON s tím, že priorita tasku se vkládá do hlavičky zprávy. Do hlavičky zprávy jsem dodala také md5 hash těla zprávy, který bude sloužit k odhalení duplikovaných zpráv. Pokud jsou všechny zasílané parametry shodné s některou ze zpráv již čekajících ve frontě, fronta tuto zprávu díky nainstalovanému balíčku sama odstraní na základě porovnání hashů v hlavičce. K odeslání zpráv je využita právě výše popsaná třída *RabbitMQ*.

Pro každý z 10 typů tasků jsem vytvořila třídy pojmenované po jejich názvech a také rozhraní *AbstractJob*, ze kterého třídy reprezentující tasky vycházejí. V tomto rozhraní jsem naimplementovala metody, pomocí kterých lze získat celý příkaz pro spuštění daných skriptů na základě dat odeslaných ve zprávě. V potomcích *AbstractJob* bylo tedy potřeba uložit pouze konstanty udávající přesný název skriptu, další informace se dají získat až po přečtení zprávy.

Pro odebírání zpráv z fronty vystačí skript, který bude jednotlivé tasky rovnou spouštět, tedy upravený původní worker. K tomu je opět využita třída *RabbitMQ* a funkce *subscribe*, které je jako parametr předáván callback. Tento návrh vychází přímo z návrhu RabbitMQ. V předávané funkci vždy bude pomocí jména tasku vytvořena instance odpovídající třídy, které v konstruktoru budou předány všechny informace z těla zprávy. Z takto vytvořeného objektu je možné získat spustitelný příkaz, který bude spouštěn php funkcí *exec*. Po spuštění bude skript workera vyčkávat a zpracovávat zprávy z fronty. Těchto workerů lze paralelně spustit několik, k tomu bude využíván bash skript.

6.1.4 Rozšíření do budoucna

Ačkoliv je vytvořená fronta zcela funkční, kromě webového rozhraní RabbitMQ, které nás informuje o množství dat ve frontě, nemáme žádné informace o době trvání nebo chybovosti tasků. Je tedy potřeba implementaci rozšířit o evidenci potřebných dat. K tomu je možné využít databázi, aniž by příliš byla omezena rychlost zpracování, jelikož bude prováděna pouze operace INSERT na tabulku bez dodatečných indexů a vkládání se tedy nebude zpomalovat přeuspořádáním indexu. Nově se také podstatně zmenší počet sloupců oproti tabulce původní, všechny potřebné informace jsou zobrazeny v tabulce 6.2.

tasks

Název	Datový typ
name	varchar(256)
sid	integer
aid	integer
is_error	integer
time_executed	timestamp
time_finished	timestamp

Tabulka 6.2: Nová tabulka *tasks*

6.2 Analýza délky běhu tasků

V nástoji BiddingManager implementace tasků byla zachována ve své původní podobě a doposud byla pouze udržována. Firma se rozhodla v rámci dalších kroků vylepšení původního návrhu o přepsání implementací tasků. K tomuto rozhodnutí došlo, jelikož v porovnání s podobnými tasky v příbuzné službě BiddingTools, je rychlost tasků BiddingManageru značně pomalejší (vztah ke službě BiddingTools je vysvětlen v části 3.1). Z důvodu omezených programátorských zdrojů přidělených k projektu se firma rozhodla přepisovat implementaci tasků postupně dle priorit, které se měly určit na základě analýzy délky jejich běhu. Byl mi tedy zadán úkol tuto analýzu zpracovat dle zadané specifikace.

6.2.1 Zadání vytvoření přehledu

Cílem bylo na základě dat z databázové tabulky *tasks* (tabulka 6.1), odhalit tasky, které běží ze všech nejdéle. Výstupní přehled měl být ve formátu xlsx souboru, kde každý druh tasku bude uložen ve svém samostatném listu. Délky běhu se dále měly rozdělit dle jednotlivých zbožových srovnávačů (Heureka.cz, Heureka.sk, Zboží.cz). Ačkoliv jsou si tasky napříč zbožovými srovnávači podobné, jejich implementace je rozdílná, jelikož využívají jiných postupů při komunikaci se zbožovými srovnávači. Kromě určení průměrné hodnoty délky běhu tasků pro každý z účtů srovnávače byl požadavek na zjištění minimální a maximální délky běhu tasků. Při vytváření analýzy nebylo požadováno vytvoření závislosti na počtu produktů a případného exponencionálního růstu, protože jejich počet je v nástroji limitován a tím pádem nedojde ani k překročení určité délky běhu, která je stále považována za akceptovatelnou. Analýza se má zaměřit spíše na odhalení tasků, které zabírají největší množství výpočetního času napříč všemi účty srovnávačů, než na odhalení tasků, kdy je výpočetní čas dlouhý jen u specifických případů. Dále měl v xlsx souboru vzniknout samostatný list pro sdružení dat a výpočet celkových průměrů mezi všemi účty srovnávačů, aby byly údaje jednoduché k porovnání.

6.2.2 Řešení vytvoření přehledu

Protože se data, která se měla zpracovávat, nacházela v jedné databázové tabulce, vytvořila jsem třídu *Task* mapující data z tabulky *tasks*. Nebylo však potřeba jí přiřadit všechny sloupce z tabulky, jelikož by pro ně nebylo využití. K vytvoření analýzy vystačily atributy popisující jméno tasku, typ zbožového srovnávače, id účtu srovnávače a čas spuštění a ukončení. Vytvořila jsem pro třídu také metodu *getExecutionTime*, která slouží pro výpočet délky běhu z atributů *timeExecuted* a *timeFinished*.

K získávání dat z databáze jsem vytvořila třídu *TaskRepository*, ve které jsem naimplementovala metodu pro vybírání záznamů v určitém časovém úseku. Databáze však obsahuje značné množství dat, a proto nebylo možné načítat všechny řádky najednou. V případě, že by byl zadán příliš velký časový úsek, mohlo by dojít k překročení kapacity paměti. Záznamy z databáze tak byly získávány po 10000 a vraceny v metodě využitím generátoru za pomoci klíčového slova *yield*. Ukázka využití *yield* v metodě je zobrazena ve výpisu 6.2.

```
$offset = 0;
$limit = 10000;
while ($rows = $this->db->query($query, $from, $to, $limit, $offset)->fetchAll())
{
   $offset += $limit;
   foreach ($rows as $row) {
      $task = Task::mapFromDibiRow($row);
      yield $task;
   }
}
```

Výpis 6.2: Kód pro vytvoření generátoru záznamů v databázi

Pro samotnou logiku seskupení a výpočtu časů jsem vytvořila třídu *TaskAnalysis*, která data za pomoci výše zmíněných tříd odebírala z databáze a připravovala pro další zpracování. Ukázka metody, ve které dochází k výpočtu průměrné, maximální a minimální časy se nachází ve výpisu A.1.

K práci se souborem jsem využila knihovnu *mk-j/php_xlsxwriter*, ta nabízí jednoduchý zápis do xlsx souborů, je potřeba jen poskytnout správně setřízené pole a odpovídající hlavičky. Posledním krokem bylo vytvoření třídy, která bude soubory prostřednictvím této knihovny vytvářet, tu jsem pojmenovala *TaskSheets*. Ve spustitelném skriptu tak stačilo vytvořit instanci této třídy, předat časový úsek a vygenerovat soubor.

6.2.3 Výsledky analýzy

Z vygenerované tabulky jsme zjistili, že nejpomalejším taskem je u všech zbožových srovnávačů výpočet optimálního CPC s průměrnou délkou běhu 20 minut. V blízké budoucnosti tedy dojde k jeho optimalizaci. Až na tasky, jako kontrola přihlašovacích údajů nebo generování sortiment reportu, které průměrně trvaly v řádech sekund, bude postupně optimalizována většina. Díky tomuto generátoru však víme, na co se máme zaměřit jako první, a také jej můžeme použít na porovnání po optimalizaci.

6.3 Migrace dat z databáze do BiddingManager API

Firma se rozhodla pro vytvoření vlastního Representational State Transfer (REST) API pro nástroj BiddingManager. Cílem této změny je vytvoření jednotného rozhraní pro práci s daty. Toto rozhodnutí také podpořil plánovaný vznik doplňku pro službu Shoptet, kde uživatelé budou mít možnost registrace do nástroje BiddingManager nebo například zobrazení statistik z nástroje. Není vyloučeno ani zpřístupnění tohoto API uživatelům. Samotné REST API bylo vyvinuto externím týmem, aby se však mohlo začít používat, muselo nejdříve dojít k přesunu dat z dosavadní databáze do API. Proto jsem dostala za úkol vytvořit migrační skript, který tento přesun provede.

6.3.1 Zadání migračního skriptu

Cílem bylo přesunout informace o uživatelích a jejich účtech na srovnávačích do vytvořeného API. Schéma migrované části databáze je zobrazeno na obrázku A.1. Informace o uživatelích měly být odeslány na koncový bod *User* a informace o účtech na srovnávačích ze všech tří databází oddělených dle zbožových srovnávačů na jeden koncový bod *Account*. Dále bylo potřeba zachovat přístupy uživatelů k účtům srovnávačů. Některé části, jako například fakturace migrovány prozatím nebudou, aby nedošlo ke ztrátě vazby mezi fakturou a uživatelem na základě jeho id, musela být vytvořena tabulka evidující nové id uživatele.

6.3.2 Řešení migračního skriptu

Migrační skript jsem rozdělila na dvě části, migraci uživatelů a migraci údajů o srovnávačích, jelikož API je uzpůsobeno na přidávání účtů srovnávačů k existujícím uživatelům.

Nejdříve jsem vytvořila skript přesunující pouze uživatele, pro ty jsme vytvořila mapovací tabulku *api_id_mapper*, která bude obsahovat původní id uživatele a nově vygenerované id z API. Z tabulky *uzivatele* jsem vybrala všechna potřebná data, která však potřebovala lehce upravit. Větší část parametrů se obešla bez úprav, ale bylo potřeba sjednotit formát telefonních čísel, protože v API se telefonní číslo validuje ohledně maximální délky devíti znaků. V databázi se nacházela telefonní čísla ve formátech jako „+420 123 456 789“ nebo „00420 123 456 789“, k jejich úpravě vystačily funkce *str_replace* a *prag_replace*. Informace o každém uživateli jsem namapovala do pole

indexovaného dle názvů parametrů v API, na rozdíl od jiných programovacích jazyků, PHP podporuje klíče v poli v podobě řetězce, nebylo tedy potřeba vytvářet slovník nebo hash tabulku, jak by tomu bylo například v jazyku C#. Takto připravená pole mohla být zakódována do formátu JSON, který je při komunikaci s API vyžadován. Za pomoci klienta Guzzle jsem pro vložení údajů uživatele do API odeslala POST požadavek se zakódovanými daty na koncový bod API *Users*. Posledním krokem bylo za pomoci klienta Guzzle zkontrolovat, zda došlo k úspěšnému vytvoření uživatele porovnáním stavového kódu odpovědi na odeslaný požadavek, konkrétně je požadován kód *201 - created*. V případě úspěchu jsem dekódovala nové id uživatele obsažené v těle odpovědi a vložila jej do vytvořené mapovací tabulky.

Poněkud složitější byla migrace srovnávačů. Informace, které API požaduje, jsou totiž rozmístěny nejen v několika tabulkách, ale také ve více databázích rozdělených dle typů zbožových srovnávačů. Nejprve tedy bylo potřeba vybrat všechna data z hlavní databáze z tabulek *srovnavace*, *eshopy*, *uzivatele* a *pristupy* najednou s využitím SQL operace JOIN. Kromě informací, které se budou zasílat do API, jsem musela navíc zjistit, v jaké databázi se nachází zbytek potřebných informací. O tom informuje sloupec *vps_id* v tabulce *srovnavace*. Se znalostí tohoto id již bylo možné sestavit název databáze, která obsahuje zbývající informace o účtu srovnávače. V cyklu jsem procházela údaje o každém účtu srovnávače s tím, že bylo potřeba připojení do zjištěné databáze a výběr zbývajících informací. Všechny informace jsem stejně jako u migrace uživatelů vložila do jednoho pole indexovaného dle názvů parametrů v API. Aby bylo možné účet srovnávače do API vložit, je potřeba vygenerovat token, který identifikuje uživatele, kterému účet srovnávače patří. K tomu bylo potřeba odeslat požadavek na přihlášení do API s e-mailem a heslem majitele a při úspěšném přihlášení byl v odpovědi obsažen požadovaný token. Dále už nic nebránilo data o účtu srovnávače zakódovat do formátu JSON a odeslat požadavek na vložení srovnávače na koncový bod *Accounts* API, ukázka odesílaných dat ve formátu JSON je ve výpisu 6.3. Na rozdíl od předchozích požadavků musel tento požadavek být odeslán s hlavičkou obsahující vygenerovaný token.

```
{
  "aggregatorId": "1",
  "aggregatorUsername": "user",
  "aggregatorPassword": "pass",
  "aggregatorShopId": "1",
  "aggregatorShopName": "shop.cz"
}
```

Výpis 6.3: Tělo HTTP požadavku pro přidání srovnávače do API

K jednomu účtu srovnávače může mít přístup více uživatelů, o kterých informuje tabulka *pristupy*. Na začátku skriptu před přesouváním jednotlivých srovnávačů jsem si uložila data z jmenované tabulky do dvourozměrného pole, kde klíči byla id srovnávačů a hodnotami byla pole se všemi id uživatelů. V tabulce *pristupy* jsou však uložena id uživatelů ze staré databáze a pro při-

dání přístupu v API jsou již potřeba nová id uživatelů. K tomu bylo možné využít dříve vytvořenou tabulku *api_id_mapper*, díky které bylo jednoduché převést stará id uživatelů na nová. Nakonec byl jen pro každý účet srovnávače a každého uživatele z odpovídajícího pole odeslán požadavek pro přidání přístupu a migrační skript byl kompletní.

Kapitola 7

Změny v obchodním modelu nástroje BiddingManager

1. 2. 2021 došlo ke změnám v obchodním modelu BiddingManageru týkajících se určování ceny za nástroj a následné fakturace. Z tohoto důvodu bylo potřeba upravit některé části.

7.1 Jednotná cena za nástroj

První změnou bylo sjednocení ceny za nástroj. Dříve se cena odvíjela od počtu produktů aktivovaných v autobiddingu, který mohli uživatelé kdykoliv navýšit, nově firma stanovila, že součástí zakoupeného balíčku bude vždy 500 produktů pro jeden účet srovnávače. Na základě těchto změn bylo mým úkolem provést úpravy, po kterých nebude uživatelům zpřístupněno nastavení velikosti balíčku a jeho velikost bude vždy nastavena na pevně daný limit.

Jelikož v době před novým obchodním modelem velikost balíčku pro autobidding mohli určovat sami uživatelé, bylo potřeba tuto hodnotu pro každý účet srovnávače ukládat do databáze. Nově však stačí uchovávat informaci o statické velikosti balíčku. Proto jsem vytvořila třídu *Constants*, do které byla uložena konstanta udávající velikost balíčku. Tato třída nám do budoucna umožní měnit a přidávat konstantní hodnoty na jednom místě. Nakonec jsem musela najít všechny výskyty načítání velikosti z databáze a nahradit je nově vytvořenou konstantou.

Na tyto změny bylo potřeba reagovat i ve webovém rozhraní, ve kterém bylo uživatelům dostupné nastavení velikosti balíčku. Všechny části týkající se tohoto nastavení byly odstraněny, což vedlo také k drobným úpravám vzhledu a rozložení. Nová velikost balíčku se propisuje do rozhraní z back-end části, upravit však bylo potřeba navíc i nápovědu, doplněnou o nové obrázky, nebo emaily doručované uživatelům. Aby byli uživatelé o proběhlých změnách informování, bylo vytvořeno vyskakovací okno shrnující všechny novinky.

7.2 Změny fakturace

Na neměnnou cenu navazuje i druhá změna, kterou je generování faktur na začátku měsíce. Jelikož uživatelé již nemají možnost s cenou manipulovat, bude možné ji vypočítat předem na celý měsíc. Při této příležitosti byla část zodpovídající za fakturace upravena a rozšířena. Níže popisuji dva úkoly, které mi byly zadány v rámci změn způsobu fakturace.

7.2.1 Nový příznak v databázi

Do faktury se uživatelům nemusí propisovat ceny za všechny účty srovnávačů, některé mohou být ve zkušebním režimu zdarma a pro některé klienty se fakturace nevytváří. Původně položky na fakturách spravovala produktová specialistka ve webovém rozhraní a musela každou z položek faktury pro daný měsíc odstranit ručně. Tento proces však bylo možné zjednodušit doplněním databázové tabulky o informaci, zda se má cena za účet srovnávače započítat do výsledné faktury. Měla jsem tedy za úkol vytvořit příznak v databázové tabulce informující o generování faktury. V návaznosti na tuto změnu jsem také měla zavést systém *Doctine migrations* pro verzování databáze, se kterým jsem se seznámila právě díky tomuto úkolu.

K rozpoznání, zda se má faktura generovat, jsem přidala do tabulky *srovnavace* sloupec *generate_invoice*, který nabývá hodnot 1 a 0. Ten se bude dále využívat u všech SQL příkazů, které budou vybírat položky k fakturaci. Při změně struktury databáze je vhodné tyto změny zaznamenávat, k tomu se dají využít takzvané migrace databáze. Jelikož BiddingManager využívá databázi produkční, testovací a sami vývojáři mohou mít své lokální databáze, je potřeba o těchto změnách zachovat přehled, jinak by aktuální kód nemusel odpovídat struktuře databáze. K vytvoření migrací byla zadána komponenta *Doctrine migrations*. Ta umožňuje vytváření tříd s migracemi pomocí příkazové řádky, stačí třídu po vytvoření doplnit o provedené změny v databázi. Ukázka takto vygenerované třídy doplněné o záznam s přidáním nového sloupce je ve výpisu 7.1. Stejně tak lze pomocí příkazové řádky změnu v databázi automaticky aplikovat či odstranit, k tomu je potřeba specifikace údajů pro databázovém připojení v konfiguračním souboru *migrations-db.php*.

```
class Version20210127132237 extends AbstractMigration
{
    /**
     * @param Schema $schema
     */
    public function up(Schema $schema)
    {
        $this->addSql("ALTER TABLE srovnavace ADD generate_invoice int(4) DEFAULT 1"
            );
    }
```

```
    /**
     * @param Schema $schema
     */
    public function down(Schema $schema)
    {
        $this->addSql("ALTER TABLE srovnavace DROP generate_invoice");
    }
}
```

Výpis 7.1: Vygenerovaná třída migrace doplněná o SQL příkaz

7.2.2 Příkaz pro manuální generování

Dalším z mých úkolů bylo vytvořit příkaz pro manuální generování faktur na základě jejich kódu. Příkaz bude užitečný pro vývojáře například při lokálním testování. Tento úkol mi byl zadán také proto, abych se seznámila s komponentou *Symfony Console*, ta slouží k vytváření příkazů pro spustitelné skripty. Oproti klasickému spouštění php skriptů na základě názvu souboru sjednocuje jejich zápis a umožňuje specifikovat také povinné i nepovinné vstupní argumenty nebo definici možností spuštění. Dále také seskupuje všechen spustitelný kód na jedno místo, a je tak velice jednoduché získat seznam všech příkazů i s jejich popisem. Hodí se využít především pro skripty, které jsou využívány opakovaně.

Pro vytvoření příkazu bylo potřeba vytvořit potomka třídy *Command* s názvem *GenerateInvoiceCommand*. V metodě *configure* se specifikuje název příkazu a argumenty, v tomto případě je potřeba povinně zadat kód faktury, která se má vygenerovat. Samotná spustitelná část se implementuje v metodě *execute*, v ní jsem z databáze vybrala záznam odpovídající zadanému kódu a pomocí příslušných existujících tříd bylo možné vygenerovat fakturu.

Kapitola 8

Znalosti získané během studia

Ačkoliv je učivo na vysoké škole spíše teoretické, v mnoha ohledech mě zvládlo připravit na absolvovanou praxi. V této kapitole je popsáno, jaké znalosti získané během studia jsem měla příležitost využít, ale také znalosti, které jsem získala až při absolvování praxe.

8.1 Uplatněné znalosti během praxe

Kromě znalostí z předmětů Programování I a II a Algoritmy I a II, které poskytují studentům nezbytný základ k programování, bych jako velice přínosné označila znalosti z předmětu Vývoj informačních systémů. Díky němu jsem se seznámila nejen s podstatou softwarových architektur, návrhovými vzory, které jsem během vývoje využívala, ale také s výhodami agilního vývoje, kterého jsem byla součástí právě během absolvování praxe. Uplatnila jsem také znalosti ohledně tvorby databázových příkazů z předmětů Úvod do databázových systému. Rovněž jsem často využívala znalosti objektově relačního mapování, se kterým jsem se poprvé setkala v předmětu Databázové a informační systémy. Ve volitelném předmětu Vývoj internetových aplikací jsem si prohloubila dovednosti ohledně tvorby webů.

8.2 Chybějící znalosti během praxe

Bohužel jsem během studia neměla šanci seznámit se s jazykem PHP, nejednalo se však o příliš velkou překážku, jelikož v mnoha ohledech si je jazyk podobný s jinými vyučovanými jazyky jako například Java. Musela jsem se také naučit pracovat s frameworkem Nette a s knihovnami, které firma běžně využívá. Poněkud náročné pro mě ze začátku bylo pracovat s verzovacím systémem Git, se kterým jsem se při studiu nesetkala.

Kapitola 9

Závěr

Výsledkem této práce bylo několik změn a nových rozšíření nasazených do každodenního provozu nástroje BiddingManager, mezi nimi například optimalizace výkonu nástroje, ale také úpravy webového rozhraní. Za poměrně krátkou dobu jsem si osvojila mnoho dovedností, jako je vývoj v týmu nebo práce s novými technologiemi, v první řadě pro mě málo známý jazyk PHP. Dále jsem také byla seznámena s fungováním zbožových srovnávačů. V neposlední řadě jsem zjistila, co všechno vývoj software obnáší, od samotné komunikace s produktovými specialisty až po samotnou implementaci. K získání všech těchto znalostí a dovedností mi velice pomohli také mí kolegové a nadřízení, na které jsem se mohla obracet se svými dotazy a nejasnostmi. Věřím, že všechny znalosti načerpané absolvováním praxe budu dále rozšiřovat v rámci profesního rozvoje, ale také při navazujícím studiu.

Jelikož se jednalo o mé první seznámení s prací software developera, nevěděla jsem, co přesně očekávat, ale byla jsem celým průběhem praxe příjemně překvapena. Absolvovanou praxi tedy nemůžu hodnotit jinak než kladně, vykonávaná práce mě bavila a přístup firmy ke mě jako zaměstnanci byl velice dobrý. Ve vývoji nástroje BiddingManager a jiných firemních aplikací budu pokračovat ve společnosti TRAYTO i nadále po dokončení této bakalářské práce.

Literatura

1. *Trayto* [online]. 2021-03-28 [cit. 2021-03-28]. Dostupné z: `https://trayto.com`.
2. *Biddingmanager.cz* [online]. 2021-03-28 [cit. 2021-03-28]. Dostupné z: `https://biddingmanager.cz`.
3. *BiddingTools.cz* [online]. 2021-03-28 [cit. 2021-03-28]. Dostupné z: `https://biddingtools.cz`.
4. *Php.net* [online]. 2021-03-28 [cit. 2021-03-28]. Dostupné z: `https://www.php.net`.
5. TATROE, Kevin; MACINTYRE, Peter. *Programming PHP*. O'Reilly Media, Inc., 2020. Version 4. ISBN 9781492054139.
6. *MariaDB* [online]. 2021-03-28 [cit. 2021-03-28]. Dostupné z: `https://mariadb.org`.
7. LOPEZ, Antonio. *Learning PHP 7*. Packt Publishing, 2016. ISBN 9781785880544.
8. CHACON, Scott; STRAUB, Ben. *Pro Git*. Apress, 2014. Version 2. ISBN 1484200772.
9. *Guzzle* [online]. 2021-03-28 [cit. 2021-03-28]. Dostupné z: `https://docs.guzzlephp.org/en/stable/`.
10. JOHANNSON, Lovisa. *The optimal RabbitMQ guide*. 2021. Version 2. ISBN 978-91-519-3115-9.

Příloha A

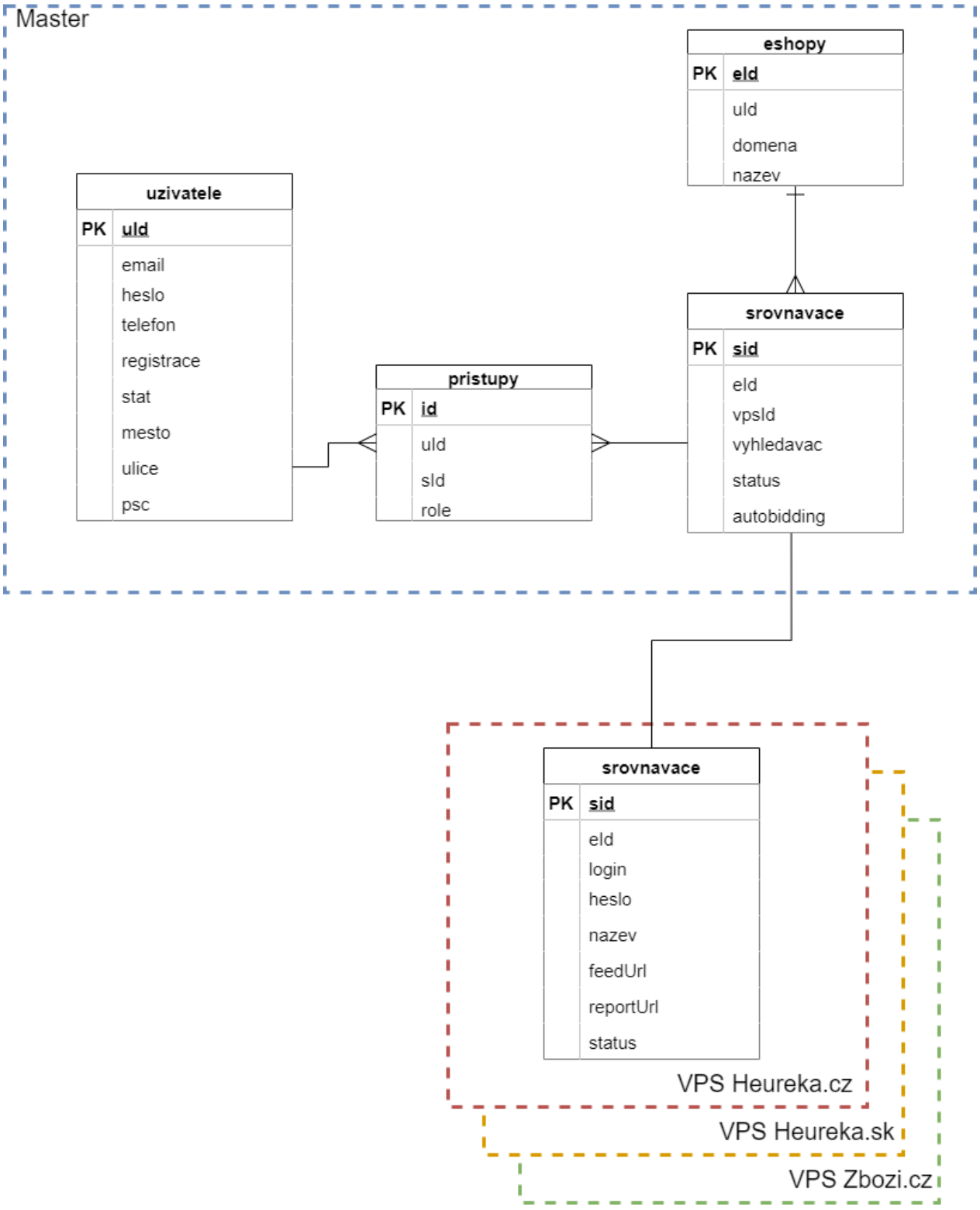
Zdrojové kódy a obrázky

```
public function getGeneralStatistics(DateTime $from, DateTime $to): array
   {
      $stats = [];
      $taskRepository = new TaskRepository($this->db);
      foreach ($taskRepository->getFinishedTasks($from, $to) as $task) {
         $name = $task->getName();
         $aggregator = $task->getAggregator();
         if (isset($stats[$name][$aggregator])) {
            $stats[$name][$aggregator]['timeTotal'] += $task->getExecutionTime();
            $stats[$name][$aggregator]['tasksAmount']++;
            if ($task->getExecutionTime() < $stats[$name][$aggregator]['timeMin'])
                {
               $stats[$name][$aggregator]['timeMin'] = $task->getExecutionTime();
            }
            if ($task->getExecutionTime() > $stats[$name][$aggregator]['timeMax'])
                {
               $stats[$name][$aggregator]['timeMax'] = $task->getExecutionTime();
            }
         } else {
            $stats[$name][$aggregator] = [
               'timeTotal' => $task->getExecutionTime(),
               'tasksAmount' => 1,
               'timeAvg' => 0,
               'timeMin' => $task->getExecutionTime(),
               'timeMax' => $task->getExecutionTime(),
            ];
```

```
        }
    }

    foreach ($stats as $name => $statsByName) {
        foreach ($statsByName as $aggregator => $taskStats) {
            $stats[$name][$aggregator]['timeAvg'] = $taskStats['timeTotal'] /
                $taskStats['tasksAmount'];
        }
    }
    return $stats;
}
```

Výpis A.1: Metoda výpočtu délek běhu tasků



Obrázek A.1: Schéma databází s uživatelskými daty